EL CICLO FOR

El primer ciclo que aprenderemos en este capítulo se llama ciclo **for**, pero para poder entenderlo mejor necesitamos ver un problema donde sea necesario utilizarlo. Imaginemos que tenemos que crear un programa para una escuela, y este programa debe sacar el **promedio** de las calificaciones para tres alumnos. Si recordamos, el promedio se calcula al sumar todas las cantidades y el resultado de la suma se divide por la cantidad de cantidades que hemos sumado. El programa es sencillo, debido a que en el salón de clases solamente hay tres alumnos.
El diagrama para resolver este programa es el que se muestra en la siguiente figura:

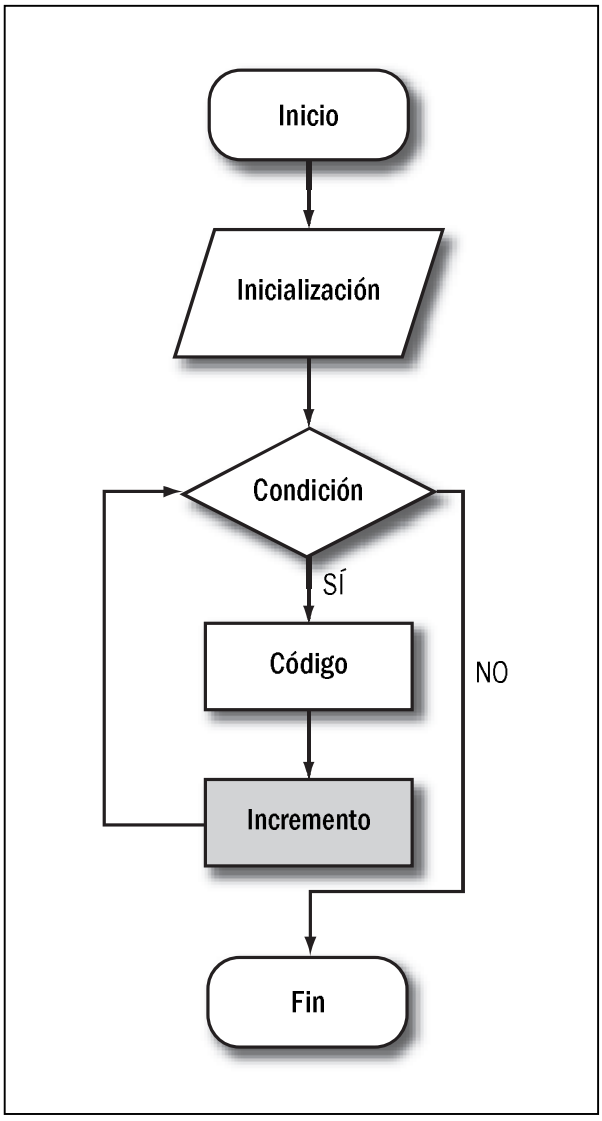

Figura 1. *Podemos observar que el programa es muy sencillo.*

El código fuente queda de la siguiente manera:

```
using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            float cal1 = 0.0f, cal2 = 0.0f, cal3 = 0.0f;
            float promedio = 0.0f;
            string valor = "";

            // Pedimos los datos
            Console.WriteLine("Dame la primera calificación:");
            valor = Console.ReadLine();
            cal1 = Convert.ToSingle(valor);

            Console.WriteLine("Dame la segunda calificación:");
            valor = Console.ReadLine();
            cal2 = Convert.ToSingle(valor);

            Console.WriteLine("Dame la tercera calificación:");
            valor = Console.ReadLine();
            cal3 = Convert.ToSingle(valor);

            // Calculamos el promedio
            promedio = (cal1 + cal2 + cal3) / 3;
```

La variable de control que usamos en el ciclo **for** puede ser declarada antes del ciclo, pero en algunos casos es posible declararla e inicializarla en la sección de inicialización del ciclo. Si la declaramos ahí, hay que tomar en cuenta que el **ámbito** de la variable es solamente el ciclo y no se conocerá por afuera de él.

```
            // Mostramos el promedio
            Console.WriteLine("El promedio es {0}", promedio);

        }
    }
}
```

Vemos que el programa es muy sencillo. Pedimos tres valores y calculamos el promedio. Al final simplemente desplegamos el resultado. No hay ningún problema con el programa, de hecho es correcto. Sin embargo, nuestro programa tiene un defecto: es poco flexible.

Ahora imaginemos que la escuela desea el promedio para un grupo con cinco alumnos. Al parecer esto no es difícil, simplemente podríamos agregar dos peticiones de variables más y modificar la fórmula. Esto no parece problemático. ¿Pero qué sucede si el grupo es de quince alumnos? Esta solución de agregar más peticiones, aunque es viable, no es cómoda. ¿Si nos piden el promedio para toda la escuela con 2500 alumnos? Entonces es evidente que no podemos seguir con este tipo de solución.

Sin embargo, de este problema podemos observar algo. Todas las peticiones de calificaciones se hacen de la misma forma y pedir las calificaciones es algo que tenemos que repetir un número de veces. Cuando tenemos código que se debe repetir un número de veces podemos utilizar el ciclo **for**. Éste nos permite repetir la ejecución de un código un número **determinado** de veces.

Antes de resolver nuestro problema con el ciclo **for**, lo primero que tenemos que hacer es aprender a utilizar el ciclo. La sentencia del ciclo se inicia con la palabra clave **for** seguida de paréntesis. Adentro de los paréntesis colocaremos expresiones que sirven para controlar el ciclo. El ciclo **for** tiene cuatro partes principales:

```
for( inicializacion; condicion; incremento)
     código
```

El ciclo **for** necesitará una o varias **variables de control**, aunque en la mayoría de los casos usaremos únicamente una variable. Veamos las diferentes partes del ciclo **for.**

En primer lugar encontramos la **inicialización**. En esta sección le damos a la variable de control su valor inicial. El valor inicial se lleva a cabo por medio de una **asignación** normal, tal y como las que hemos trabajado. La segunda sección lleva una **condición** en forma de una expresión relacional. Ésta nos sirve para controlar cuándo termina el ciclo y generalmente aquí indicamos la cantidad de vueltas que da el ciclo.

Luego tenemos el **código**. En esta sección colocamos la parte del código que deseamos que se repita. Esta sección puede ser una sentencia o un bloque de código. Por

último, se ejecuta la sección del **incremento**. Aquí indicamos cómo se modificará el valor de la variable de control para cada vuelta del ciclo.
Cuando se ejecuta el ciclo **for**, lo primero que se lleva a cabo es la inicialización y luego la condición. Si la condición es verdadera, entonces se pasa al código y después al incremento. Seguido del incremento vamos nuevamente a la condición y se repite el proceso. Si la condición no se cumple, entonces decimos que el ciclo finaliza, se salta al código y continúa la ejecución del programa con lo que sea que encontremos después del ciclo.
Esto lo podemos apreciar mejor en el siguiente diagrama de flujo.

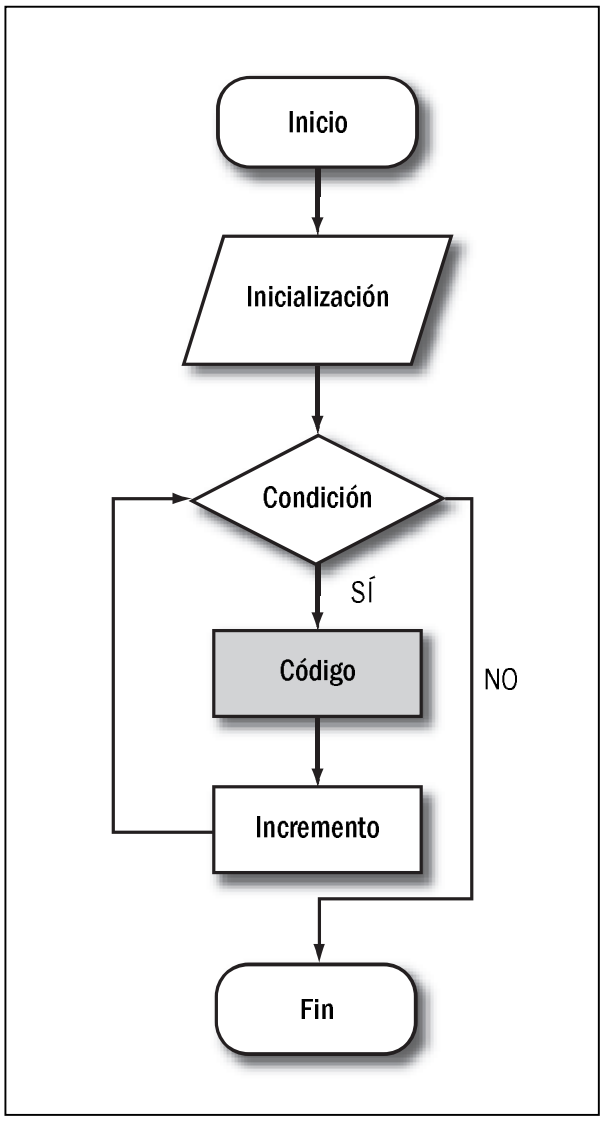

Figura 2. *Este diagrama de flujo nos muestra los pasos de la ejecución del ciclo* ***for.***

El poder comprender el ciclo **for** requiere de mucha experimentación, por lo que haremos varias pruebas para conocer sus características principales.
Empecemos con un programa sencillo. Usaremos el ciclo para mostrar un mensaje, que únicamente imprimirá el valor de la variable de control del ciclo. Para saber cuál es el trabajo que realiza el ciclo, colocaremos un mensaje antes y después del ciclo.

Para iniciar con facilidad, queremos que el ciclo se lleve a cabo diez veces. Primero veamos el programa y luego analicemos cada una de las partes del ciclo.
Nuestro programa queda de la siguiente forma:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            int n = 0;  // variable de control

            Console.WriteLine("-- Antes del ciclo --");

            for (n = 1; n <= 10; n = n + 1)
                Console.WriteLine("{0}", n);

            Console.WriteLine("-- Después del ciclo --");
        }
    }
}
```

Antes de ejecutar el programa, veamos cómo está construido. En primer lugar declaramos nuestra variable de control. Ésta se llama **n**. El programa imprimirá los números del **1** al **10** y **n** cambiará su valor según se repita el ciclo.
Adentro del ciclo for encontramos inmediatamente que a **n** se le asigna el valor **1.** Esto quiere decir que nuestro conteo empezará con el número **1**. Si deseáramos iniciar en otro valor, en esta parte es dónde lo asignaríamos. Luego tenemos la condición. La condición limita hasta donde contaremos con **n**. En este caso es hasta **10.** Mientras **n** tenga un valor menor o igual a **10** el mensaje se muestra.
Como en este caso **1 <= 10** se cumple, ya que **n** vale **1**, entonces el mensaje se escribe. El mensaje es muy sencillo ya que simplemente muestra el valor contenido en la variable **n**. Después de esto, vamos al incremento. El incremento aumenta el valor

de **n** en uno. Después del primer incremento **n** valdrá **2**. Esto se repite y se incrementa en **1** el valor de **n** cada vez que se repite el ciclo. En el último incremento, **n** tendrá el valor de **11** y en este caso la condición ya no se cumplirá y se termina con el ciclo. Veamos la ejecución del programa y observaremos cómo **n** cambia de valor y efectivamente el ciclo se lleva a cabo solamente **10** veces.

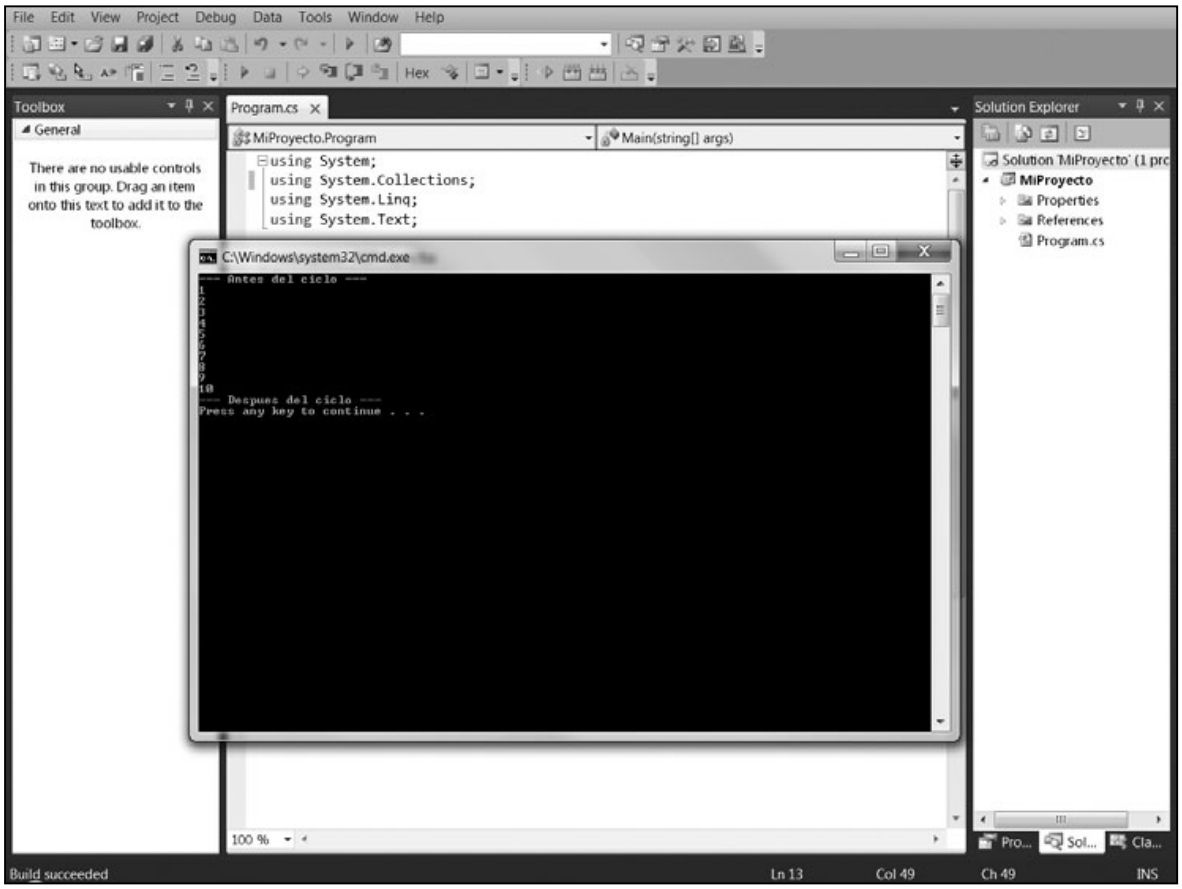

***Figura 3.** Aquí podemos observar cómo el ciclo se repitió y el valor de n fue modificado en cada vuelta.*

El valor de inicio

Ahora ya podemos empezar a experimentar con el ciclo. Lo primero que haremos será colocar el valor de inicio del ciclo y ver cómo se modifica la ejecución del

En C# veremos que en muchos casos tenemos estructuras con índice cero. Esto quiere decir que su primer elemento se considera en la posición **0**, no en la posición **1**. Aunque nos parezca extraño empezar a contar desde cero, en la computadora es muy común que esto ocurra, por lo que nos conviene acostumbrarnos a hacer ciclos de esta forma.

programa. Esto es útil ya que no siempre es necesario empezar a contar a partir de **1**, a veces necesitamos empezar a contar a partir de otros números.

Supongamos que ahora tenemos que contar del **3** al **10**. Para lograr esto, simplemente modificamos el valor de inicio en la sección de inicialización del ciclo:

```
for (n = 3; n <= 10; n = n + 1)
    Console.WriteLine("{0}", n);
```

Ejecutemos el programa.

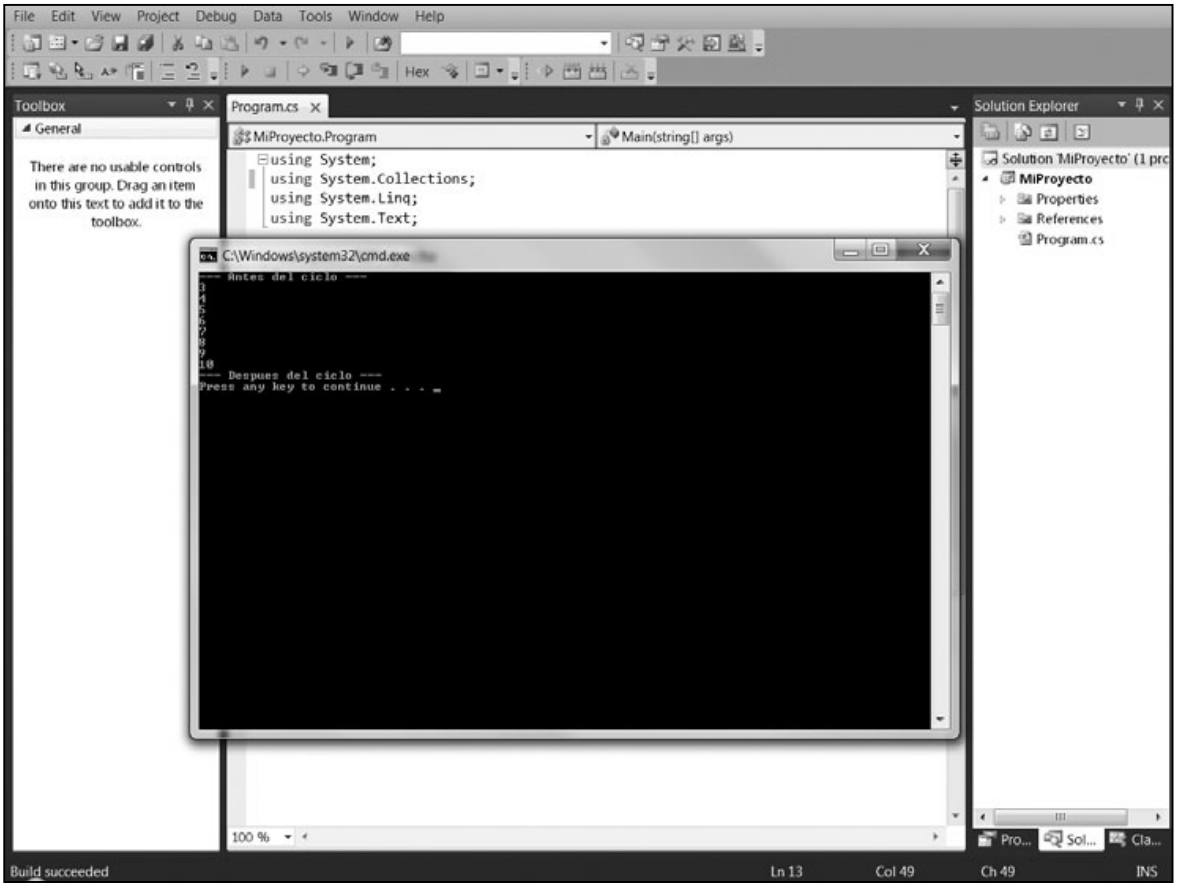

Figura 4. *Éste es el resultado. Efectivamente, la variable de control empieza con 3. Desde luego, el número de repeticiones es menor.*

Es posible que la variable de control tenga un valor negativo en la inicialización. Esto es útil cuando necesitamos contar desde un número negativo. Modifiquemos el código del ciclo for de la siguiente manera:

```
for (n = -10 n <= 10; n = n + 1)
    Console.WriteLine("{0}", n);
```

La ejecución nos da el resultado que observamos en la siguiente figura.

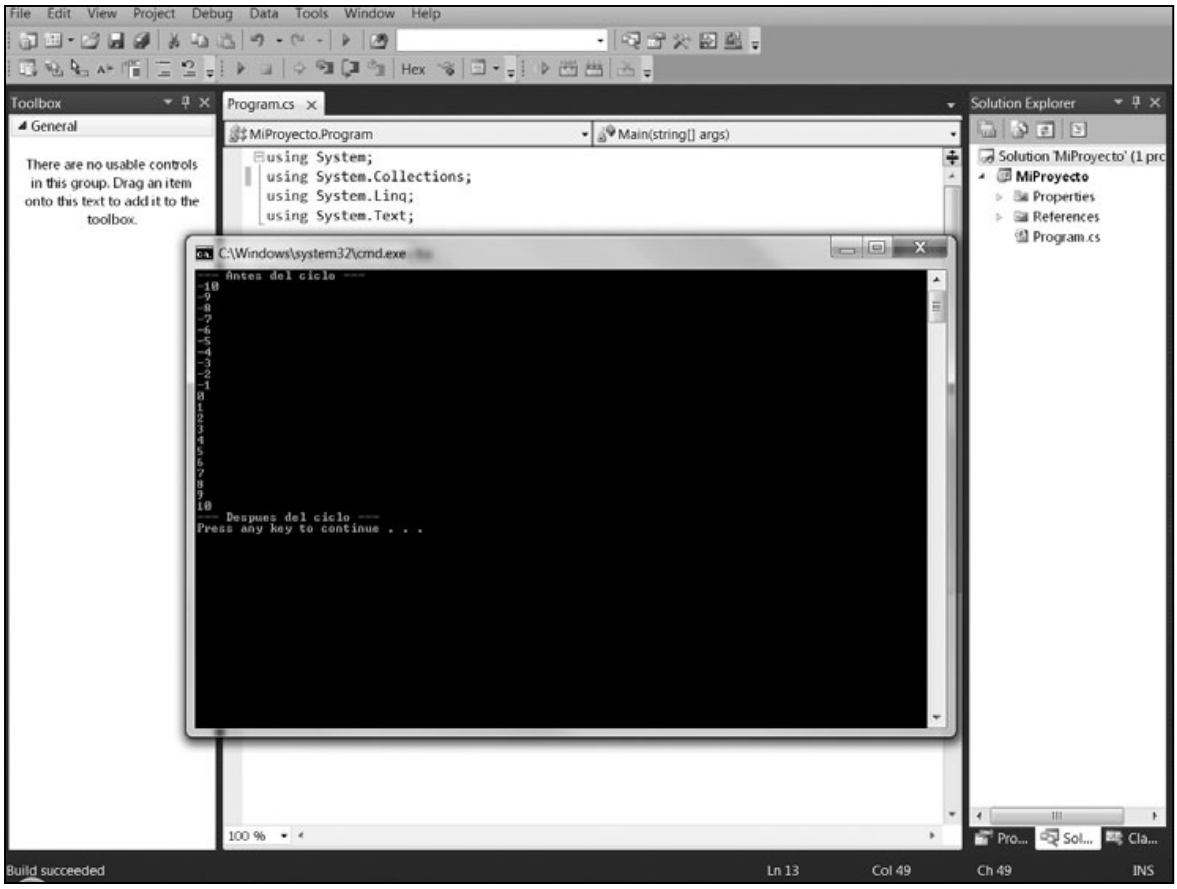

Figura 5. *Podemos ver que el recorrido inicia en -10. No hay que pasar por alto que la variable de control también pasa por el valor de cero.*

El límite de conteo del ciclo

También podemos tener un control de hasta qué número contar. Veamos nuestro código para realizar el conteo del 1 al 10.

```
for (n = 1; n <= 10; n = n + 1)
    Console.WriteLine("{0}", n);
```

En este ejemplo pudimos ver que el límite de conteo está siendo controlado en la condición y que no hay una única forma de escribirla. Podemos ver que la siguiente condición **n <= 10** también podría ser escrita como **n < 11** sin que se afecte en lo màs mínimo la ejecución del ciclo.

```
for (n = 1; n <11; n = n + 1)
    Console.WriteLine("{0}", n);
```

Veamos el resultado de este cambio en la figura a continuación.

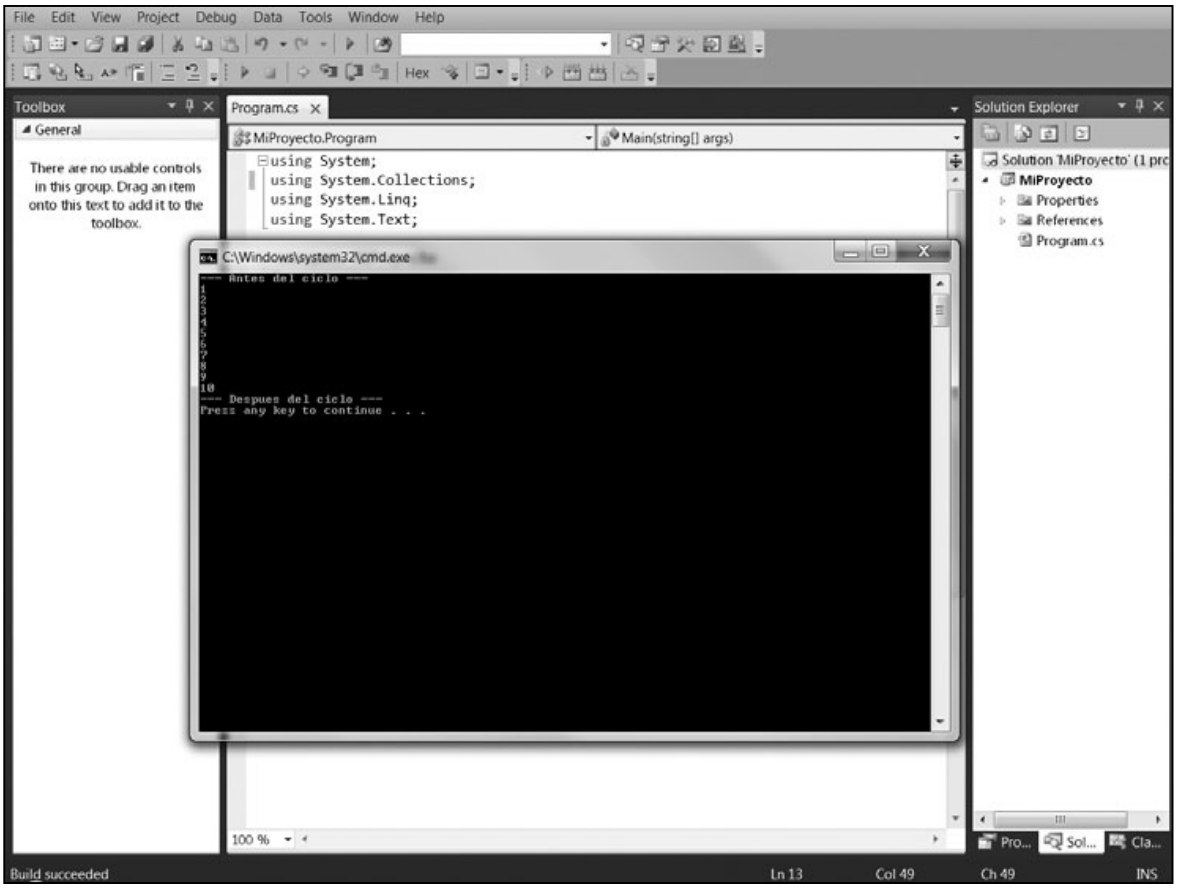

Figura 6. *Podemos observar que aunque la condición es diferente, aún se cumple lo que deseamos.*

Esto hay que tenerlo en cuenta para evitar confusiones en el futuro, en especial cuando veamos código de otras personas que utilicen un estilo diferente al nuestro. Ahora modifiquemos el ciclo para que cuente del **1** al **15**, en este caso usaremos el operador **<** en lugar de **<=.**

```
for (n = 1; n < 16; n = n + 1)
    Console.WriteLine("{0}", n);
```

Uno de los problemas más comunes con el ciclo **for** es equivocar la cantidad de repeticiones necesarias. Algunas veces sucede que el ciclo da una vuelta más que las que deseamos. Si esto ocurre, lo primero que debemos revisar es la condición. Una condición mal escrita hará que no se produzcan las repeticiones deseadas. Otro problema puede ser el colocar **;** al final de for.

Ejecutemos el programa y veamos su comportamiento.

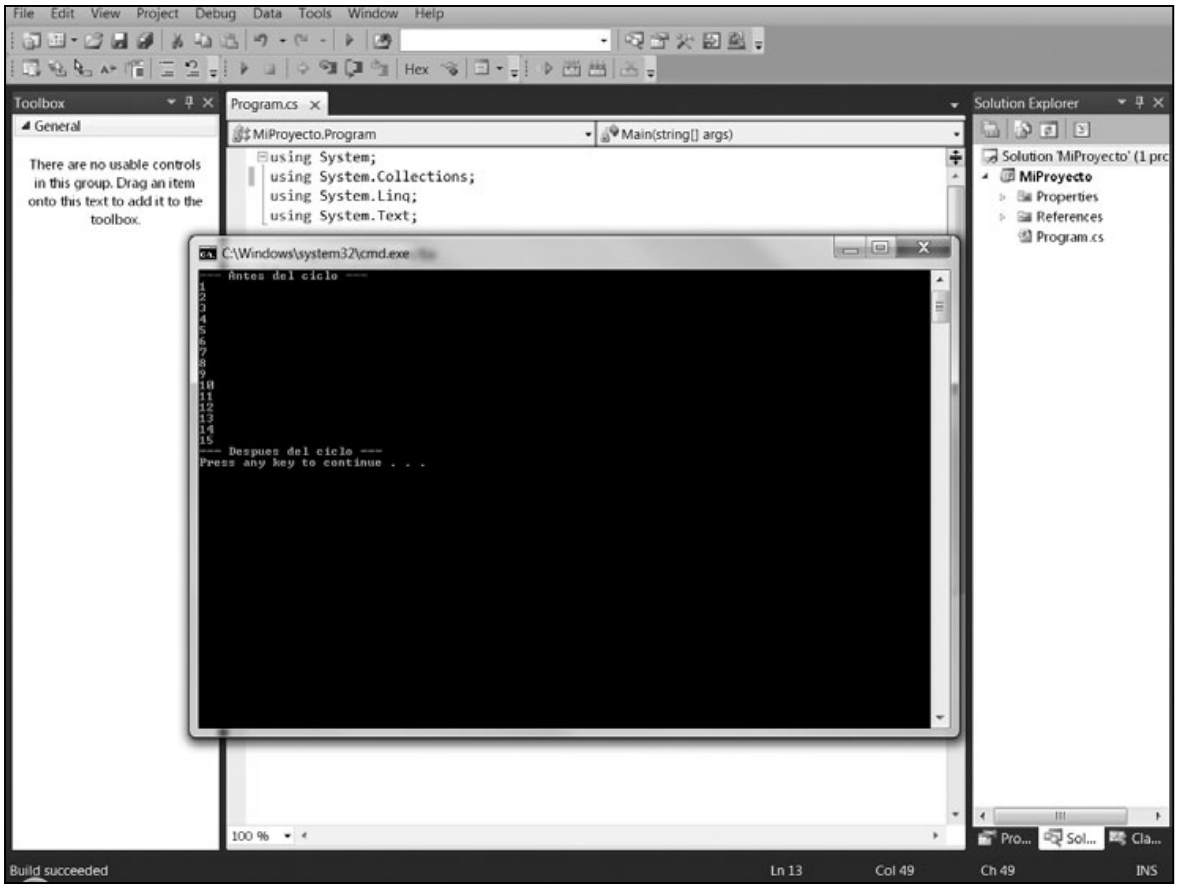

Figura 7. *En este caso hemos incrementado el rango del ciclo.*

Control del incremento

Ahora que ya sabemos cómo colocar el rango del ciclo **for** desde su valor de inicio hasta el valor al que contará, podemos empezar a aprender cómo hacer uso del incremento. Nuestro incremento ha sido de uno en uno. Sin embargo, podemos hacer que el conteo sea de dos en dos, de tres en tres o de cualquier otro valor.

Para lograr esto, simplemente tenemos que modificar el incremento e indicar cómo se incrementaría nuestra variable de control. El valor del incremento puede ser un valor colocado explícitamente o el valor que se encuentra adentro de una variable. Por ejemplo, hagamos que nuestro ciclo avance de dos en dos.

```
for (n = 1; n <16; n = n + 2)
                Console.WriteLine("{0}", n);
```

Como vemos, ahora hemos colocado **n = n + 2**. Con esto indicamos que se sumará **2** al valor de **n** con cada vuelta del ciclo. Esto es más fácil de entender si vemos la ejecución del programa y vemos el desplegado de los valores de **n**.

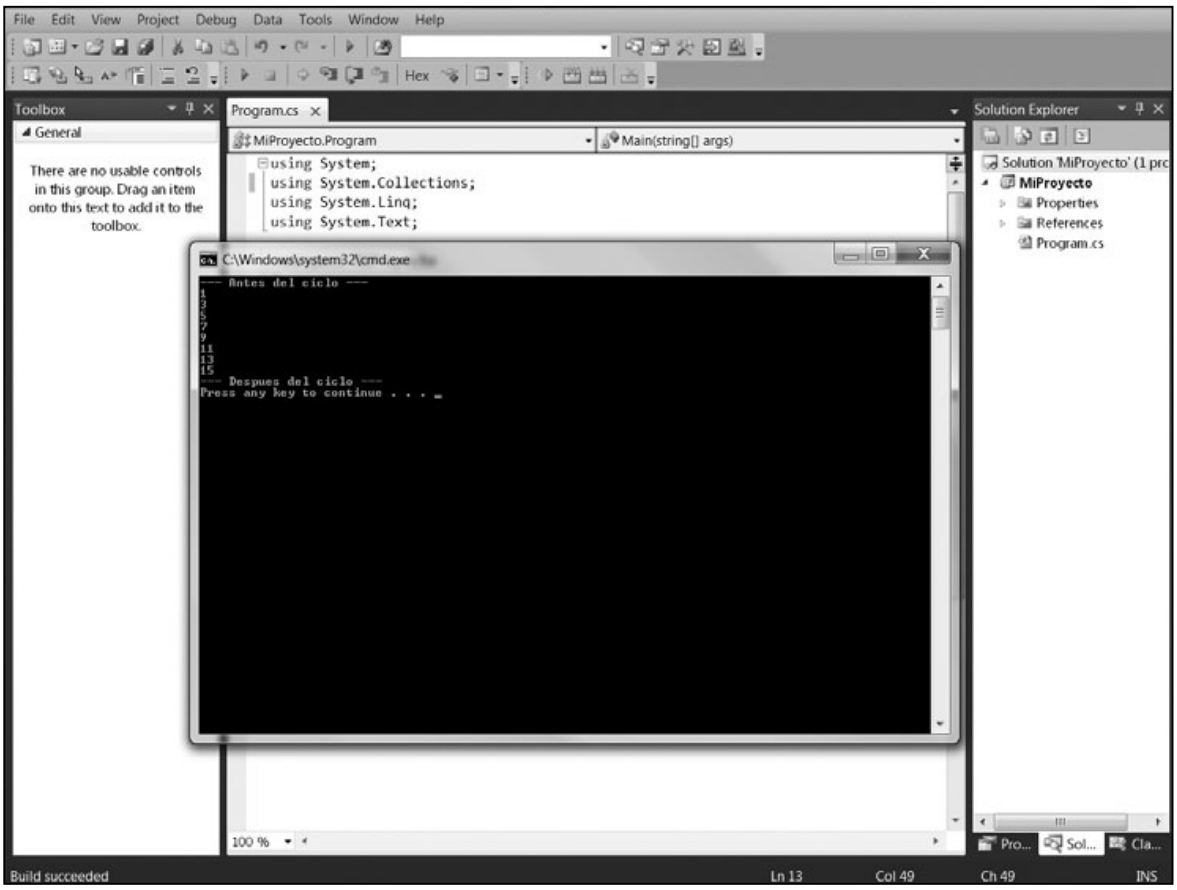

Figura 8. *Podemos observar que efectivamente el valor de n se incrementa de dos en dos por cada vuelta del ciclo.*

Los ciclos que hemos utilizado siempre han contado de manera **progresiva**, es decir del menor valor al valor más alto, pero también es posible hacer un ciclo **regresivo**. Por ejemplo, podríamos hacer que el ciclo cuente del 10 al 1. Para esto necesitamos modificar no solamente el incremento sino que también es necesario colocar las expresiones correctas en la inicialización y la condición.

Veamos cómo lograr esto con la siguiente sentencia:

```
for (n = 10; n>=1; n = n - 1)
```

Si llevamos a cabo la declaración e inicialización de una variable adentro del bloque de código de un ciclo, la variable será inicializada cada vez que se repita el ciclo y no conservará su valor entre vueltas del ciclo. Este detalle también nos puede llevar a errores de lógica, por lo que hay que decidir bien dónde se declara la variable.

```
Console.WriteLine("{0}", n);
```

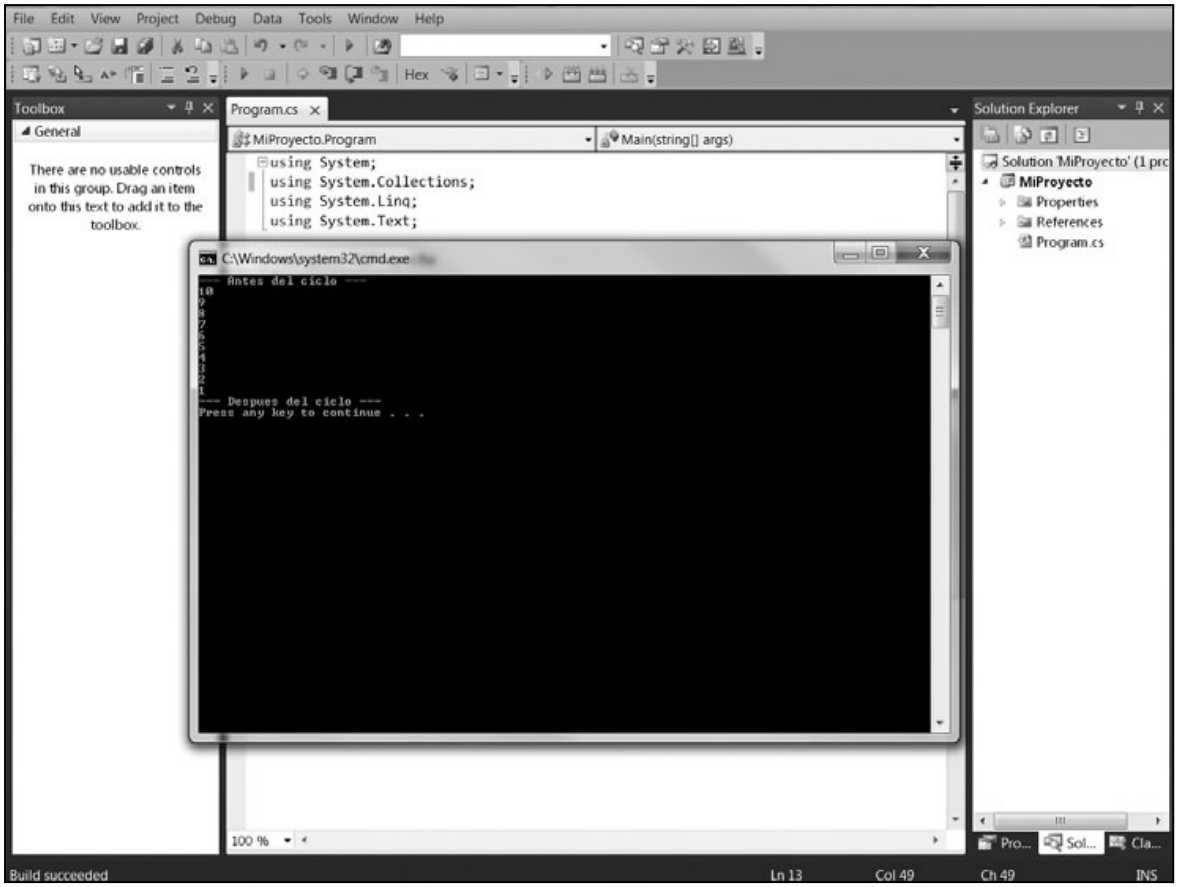

Figura 9. *Aquí podemos observar cómo el valor de n se decrementa de uno en uno.*

El contador y el acumulador

Ahora aprenderemos dos conceptos nuevos. Existen dos categorías de variables dependiendo de cómo guardan la información, en primer lugar tenemos el **contador**. El contador es una variable que será incrementada o disminuirá su valor de uno en uno. En la mayoría de los ejemplos anteriores **n** ha funcionado como contador. Por su parte, el **acumulador** es una variable que puede incrementar o disminuir su valor en cualquier número.

Aún no hemos hecho uso de un acumulador, pero lo veremos cuando debamos resolver el problema de los promedios de los alumnos.

Por el momento, veamos un pequeño ejemplo de estas dos clasificaciones de variables explicadas hasta aquí:

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            int n = 0;  // variable de control
            int contador = 0, acumulador = 0;

            Console.WriteLine("-- Antes del ciclo --");

            for (n = 10; n >= 1; n = n - 1)
            {
                contador = contador + 1;
                acumulador = acumulador + contador;
             Console.WriteLine("{0}, {1}", contador, acumulador);
            }

            Console.WriteLine("-- Después del ciclo --");
        }
    }
}
```

Hemos creado dos variables: **contador** y **acumulador**. La variable **contador** incrementa su valor de uno en uno. Por su parte, la variable **acumulador** incrementará el valor en base al número contenido en **contador.**

A continuación, para comprender mejor lo explicado, ejecutemos el programa y veamos el funcionamiento de las dos variables:

Si llevamos a cabo la declaración de una variable adentro del bloque de código del ciclo, ésta únicamente puede ser usada en ese ciclo. Si tratamos de utilizarla después del ciclo, no será reconocida por el programa. Si éste fuera el caso, lo mejor es declarar la variable al inicio de la función para que sea conocida en ella.

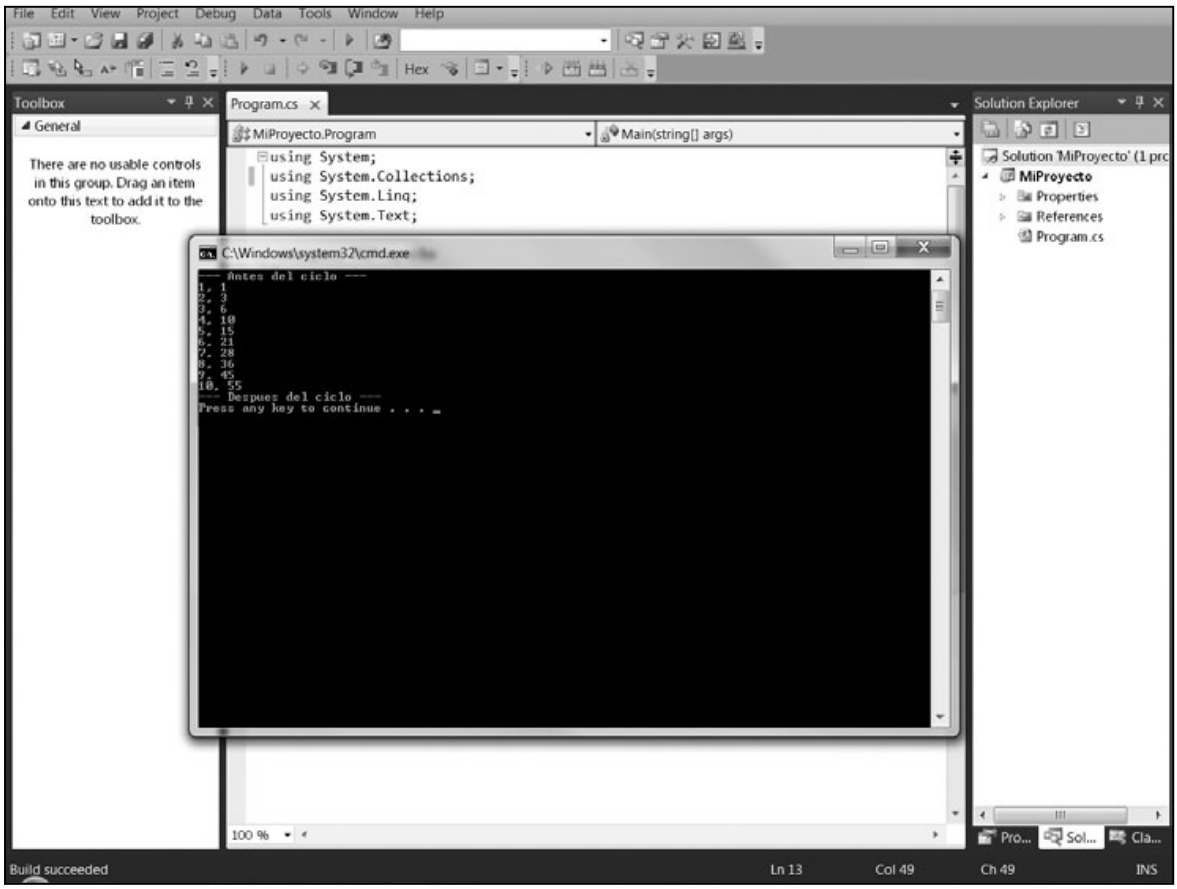

Figura 10. *Vemos cómo cambian los valores de las variables con cada vuelta del ciclo.*

Incrementos y decrementos

Cuando trabajamos el C# con el ciclo **for** es muy común que tengamos que incrementar o disminuir siempre de uno en uno. Para facilitarnos esto, tenemos operadores nuevos que son el **operador de incremento** y el **operador de decremento.**

OPERADOR	SIGNO
Incremento	++
Decremento	—

Tabla 1. *Este tipo de operadores nos permiten cambiar el valor de la variable en 1.*

Estos operadores pueden ser usados como **sufijos** o **prefijos** en la variable. En el caso de los sufijos lo escribimos como **variable++** y para el prefijo como **++variable.** El valor contenido en la variable en ambos casos se incrementará en uno. Lo que cambia es cómo se evalúa la expresión que contienen los operadores.

Cuando tenemos el caso del sufijo, se evalúa la expresión con el valor actual de la variable y luego se incrementa a la variable. En el caso del prefijo, se incrementa primero el valor de la variable y posteriormente se evalúa la expresión.

Quizás esto no quede muy claro en la primer explicación, pero no debemos preocuparnos por ello. Para poder entender esto, veremos un ejemplo a continuación:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            int numero = 5;

            Console.WriteLine("Valor inicial {0}", numero);

            // incrementamos
            numero++;

         Console.WriteLine("Después del incremento {0}", numero);

            // Decrementamos
            numero--;

         Console.WriteLine("Después del decremento {0}", numero);

            // Incremento en la sentencia
    Console.WriteLine("Incremento en la sentencia {0}", numero++);

         Console.WriteLine("Valor después de la sentencia {0}",
               numero);

            // Incremento en la sentencia como prefijo
    Console.WriteLine("Incremento en la sentencia {0}", ++numero);

    Console.WriteLine("Valor después de la sentencia {0}",
               numero);
        }
    }
}
```

Podemos ejecutar el código escrito y ver qué es lo que ha ocurrido, y la razón por la que obtenemos esos valores desplegados.

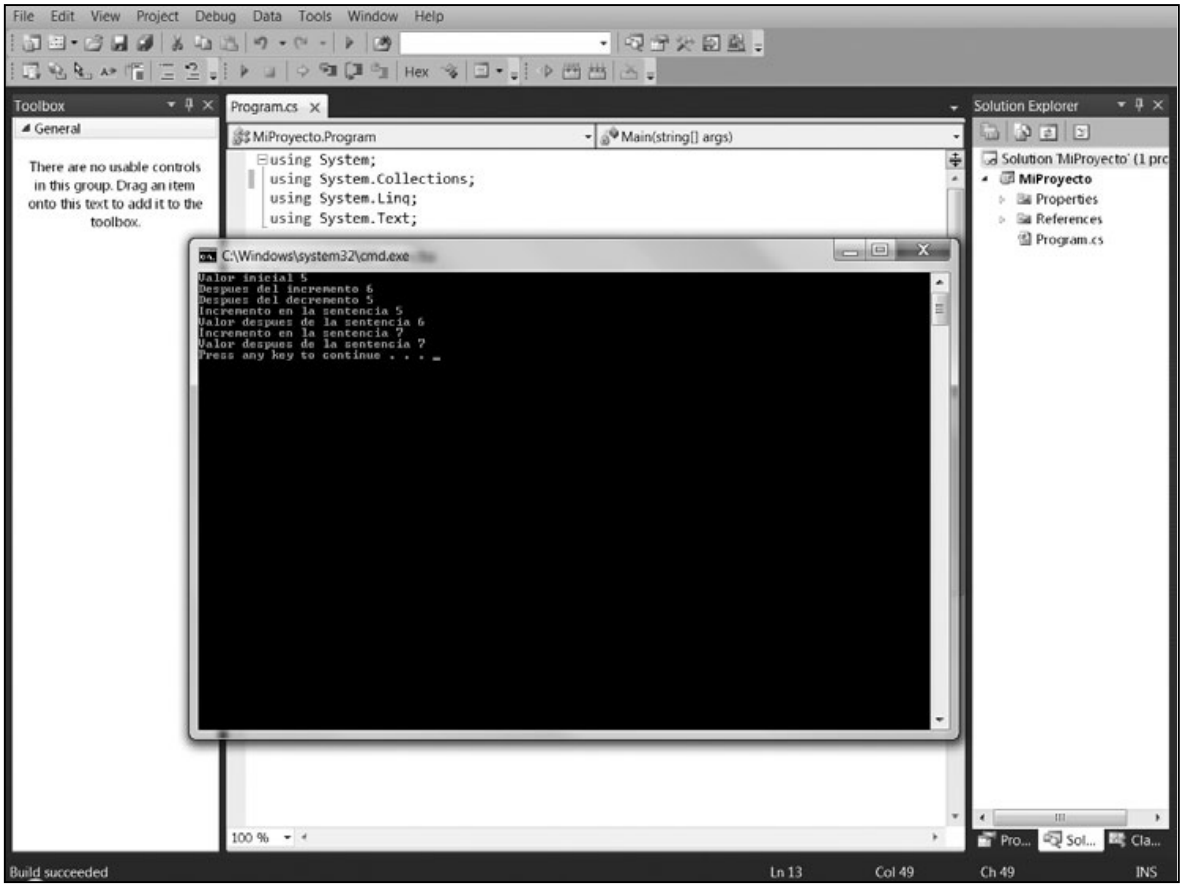

Figura 11. *Aquí vemos los valores obtenidos por los operadores y la diferencia en usarlos como sufijos o prefijos.*

Empezamos con una variable llamada **numero** a la que le asignamos el valor inicial de **cinco**. Desplegamos su valor en la pantalla. Luego usamos el operador e incrementamos el valor de la variable. En este caso vale **6**, ya que el incremento es en uno. Esto lo comprobamos al imprimir el valor de la variable. Después llevamos a cabo el decremento. En este caso, **numero** guarda el valor de **5**, ya que lo hemos decrementado en uno. La impresión del mensaje nos muestra que estamos en lo correcto.
Ahora que ya hemos visto cómo funciona, podemos tratar de colocar el operador adentro de una sentencia. La sentencia será el mismo mensaje que deseamos mostrar con el fin de observar cómo se evalúa y las diferencias entre el uso como sufijo y prefijo. No olvidemos que el último valor en **numero** es **5**. Empezamos por colocar el incremento adentro de la sentencia y lo usamos como sufijo. Esto incrementa el valor a **6**, pero si vemos la ejecución del programa, el valor que aparece es **5**. ¿Por qué? La razón ya la conocemos: cuando se usa como sufijo, se evalúa la expresión con el valor actual y luego se incrementa. La siguiente impresión del valor de la variable nos muestra cómo efectivamente la variable sí fue incrementada después de la sentencia anterior. Ahora podemos experimentar con el sufijo. Nuestro valor actual es de **6** e ingresamos la

sentencia. En este caso, primero se incrementa el valor y tenemos **7** en la variable **número** y luego la imprimimos. Por eso, en este caso obtenemos **7** en la pantalla.
Con esto ya hemos visto el comportamiento de estos operadores y podemos integrarlos en nuestro ciclo for. En el caso de incremento:

```
for (n = 1; n <16; n++)
    Console.WriteLine("{0}", n);
```

O para decrementar decrementar su valor:

```
for (n = 10; n>=1; n–)
    Console.WriteLine("{0}", n);
```

Existen otros operadores que también podemos utilizar cuando deseamos calcular un valor con la variable y guardar el mismo valor en la variable.

OPERADOR	EJEMPLO	EQUIVALE A
+=	numero+=5	numero=numero+5
-=	numero-=5	numero=numero-5
*=	numero*=5	numero=numero*5
/=	numero/=5	numero=numero/5

Tabla 2. *Estos operadores también nos permiten modificar el valor de la variable.*

Es bueno conocer estos operadores ya que los veremos y deberemos utilizar frecuentemente en muchos programas de cómputo. Por ejemplo, para hacer el incremento de cinco en cinco en un ciclo for, realizamos lo siguiente:

```
for (n = 1; n <16; n+=5)
    Console.WriteLine("{0}", n);
```

Ejemplos con el ciclo for

Ahora ya podemos resolver el problema sobre el cálculo del promedio. Sabemos que es lo que se debe repetir y la cantidad de veces que debemos hacerlo. En este caso, lo que se debe repetir es pedir la calificación del alumno y hacer una suma de estas calificaciones. La cantidad de veces que se tiene que repetir depende de la cantidad de alumnos. Esta suma de las calificaciones puede hacerse con la variable acumulador. El promedio final se calcula dividiendo el acumulador entre la cantidad de

alumnos y éste se presenta en la pantalla. Veamos el diagrama de flujo de nuestro algoritmo para resolver este problema.

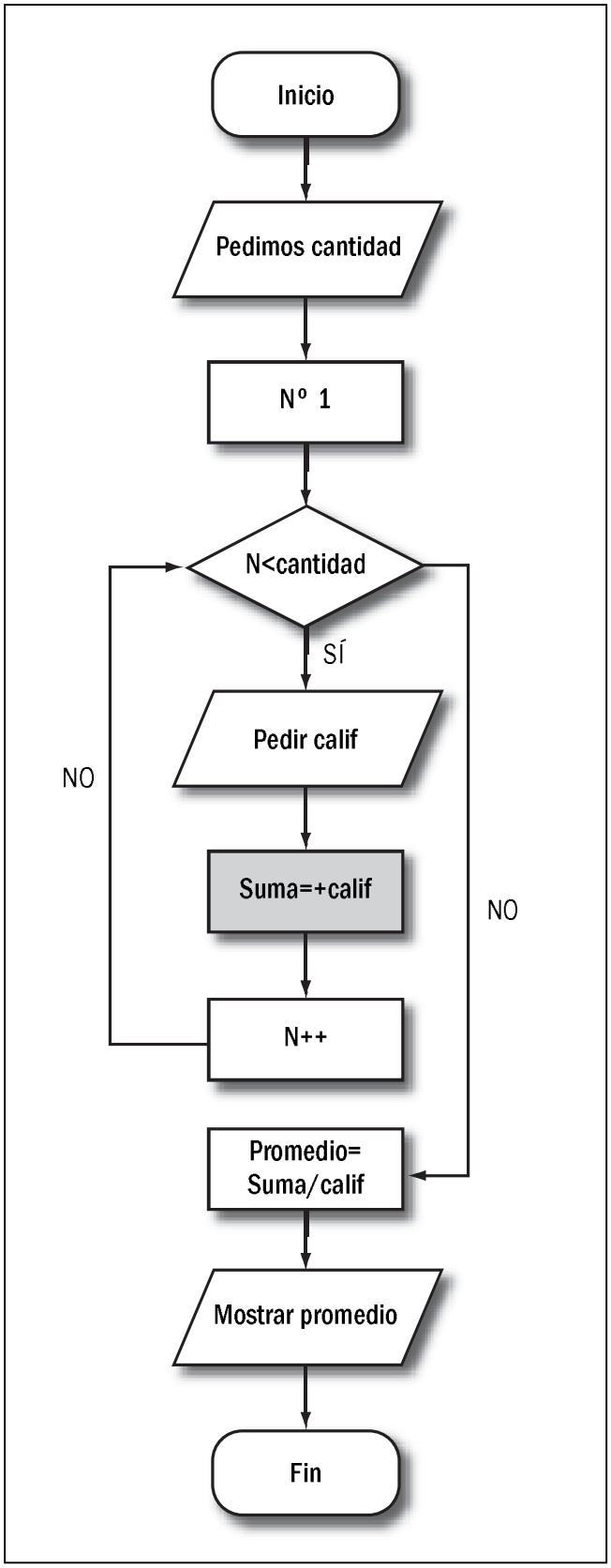

Figura 12. *Podemos observar fácilmente el ciclo for y lo que se realiza adentro de él.*

Ahora podemos ver cómo queda el código del programa:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            int n = 0;  // Control ciclo
            int cantidad;   // Cantidad alumnos
            float calif = 0.0f; // Calificación del alumno
            float suma = 0.0f;  // Sumatoria de calificaciones
            float promedio = 0.0f;   // Promedio final
            string valor = "";

            Console.WriteLine("Dame la cantidad de alumnos:");
            valor = Console.ReadLine();
            cantidad = Convert.ToInt32(valor);

            // Ciclo para la captura de calificaciones
            for (n = 1; n <= cantidad; n++)
            {
            Console.WriteLine("Dame la calificacion del alumno");
                valor = Console.ReadLine();
                calif = Convert.ToSingle(valor);

                // Llevamos a cabo la suma de calificaciones
                suma += calif;

            }

            // Calculamos el promedio
            promedio = suma / cantidad;
```

```
            // Mostramos el promedio
            Console.WriteLine("El promedio es {0}",promedio);

        }
    }
}
```

Si ejecutamos el programa obtenemos la salida que se muestra en la siguiente figura.

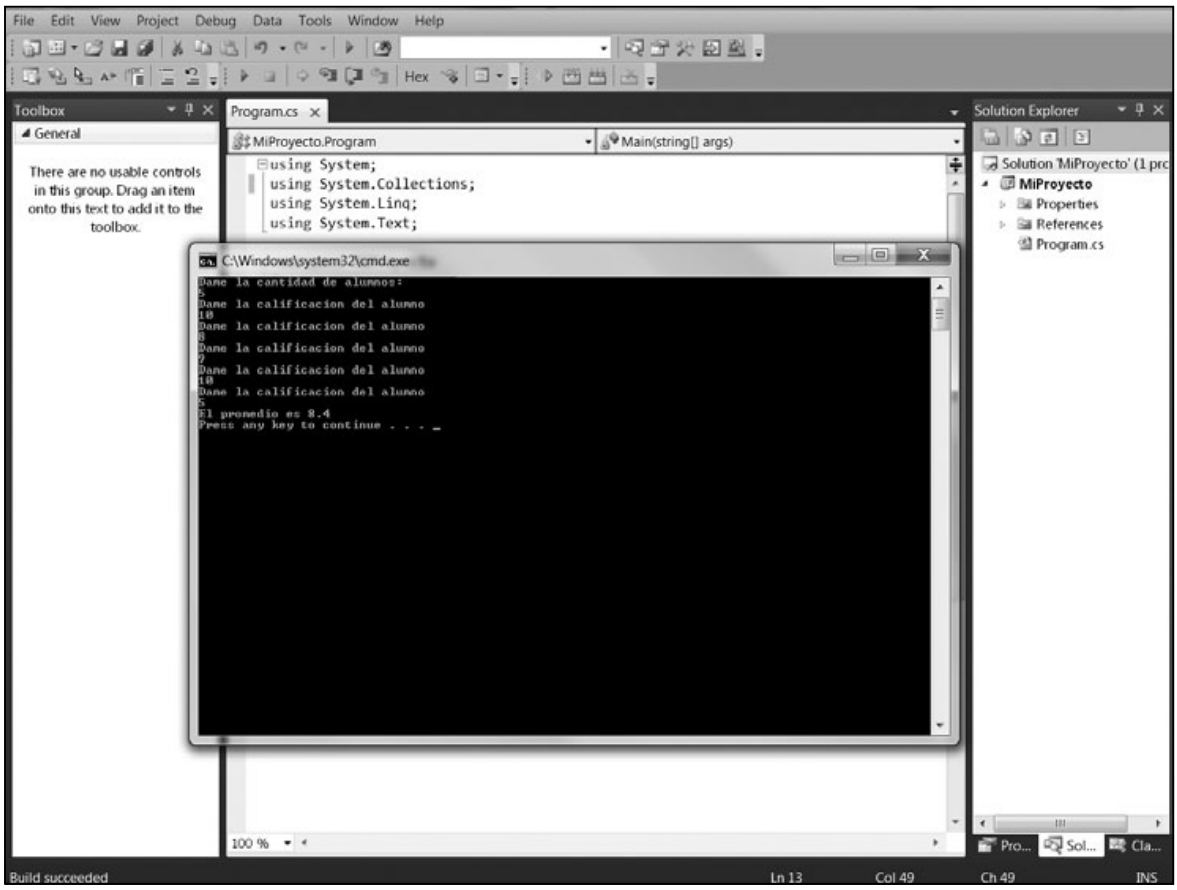

Figura 13. *Vemos cómo la petición de la calificación se repite el número de veces indicado.*

Un punto importante sobre este programa, en comparación al primer intento que hicimos al inicio del capítulo, es que puede funcionar con cualquier cantidad de alumnos. No importa si son 5, 10 ó 5000, el programa llevará a cabo su cometido. Vemos cómo el uso del ciclo no solamente nos da flexibilidad, sino que también evita que tengamos que repetir mucho código.

Veamos otro ejemplo. Podemos utilizar el ciclo **for** en cualquier problema que necesite algo que se repita: una operación, un conteo, algún proceso. Pero también el número de repeticiones debe ser conocido, ya sea por medio de un valor colocado explícitamente o un valor colocado adentro de una variable.

Ahora tenemos que calcular el **factorial** de un número. Por ejemplo, el factorial de **5** es **5*4*3*2*1** que da **120**. De igual forma, debemos calcular el valor factorial de cualquier número dado por el usuario. Si observamos el problema y cómo se resuelve, podemos apreciar que hay un ciclo y este ciclo es con conteo regresivo. Podemos resolver fácilmente con un ciclo **for** que lleve a cabo la multiplicación con la variable de control como producto.

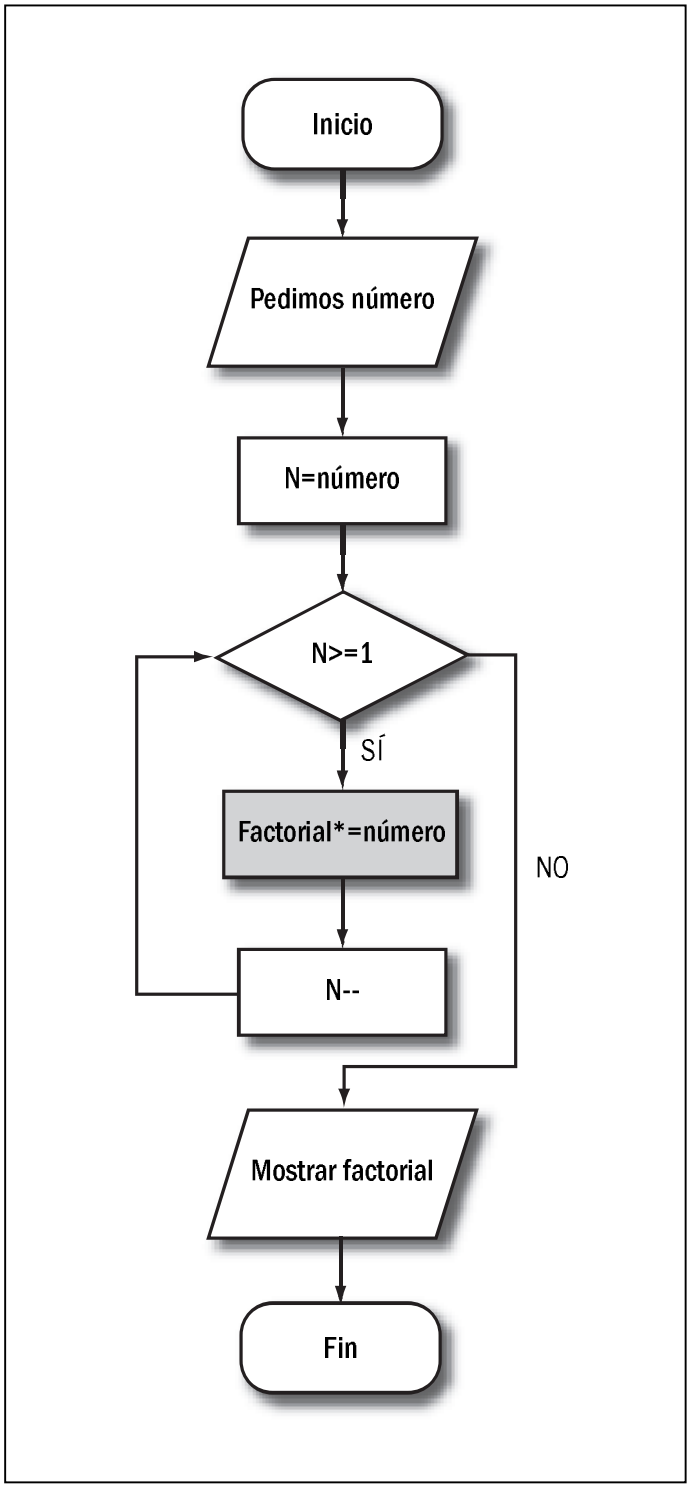

Figura 14. *Éste es el diagrama de flujo del algoritmo para calcular el factorial de un número.*

El código del programa queda de la siguiente forma:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            int n = 0;  // Variable de control
            int numero = 0; // Número al que sacamos factorial
            int factorial = 1;  // Factorial calculado
            string valor = "";

            // Pedimos el numero
            Console.WriteLine("Dame el número al que se le saca el
              factorial:");
            valor = Console.ReadLine();

            numero = Convert.ToInt32(valor);

            // Calculamos el factorial en el ciclo
            for (n = numero; n >= 1; n–)
                factorial *= n;

            // Mostramos el resultado
            Console.WriteLine("El factorial de {0} es {1}",numero,
              factorial);
        }
    }
}
```

Cuando ejecutamos el programa, nos encontramos con que podemos llevar a cabo el cálculo factorial sin problema alguno.

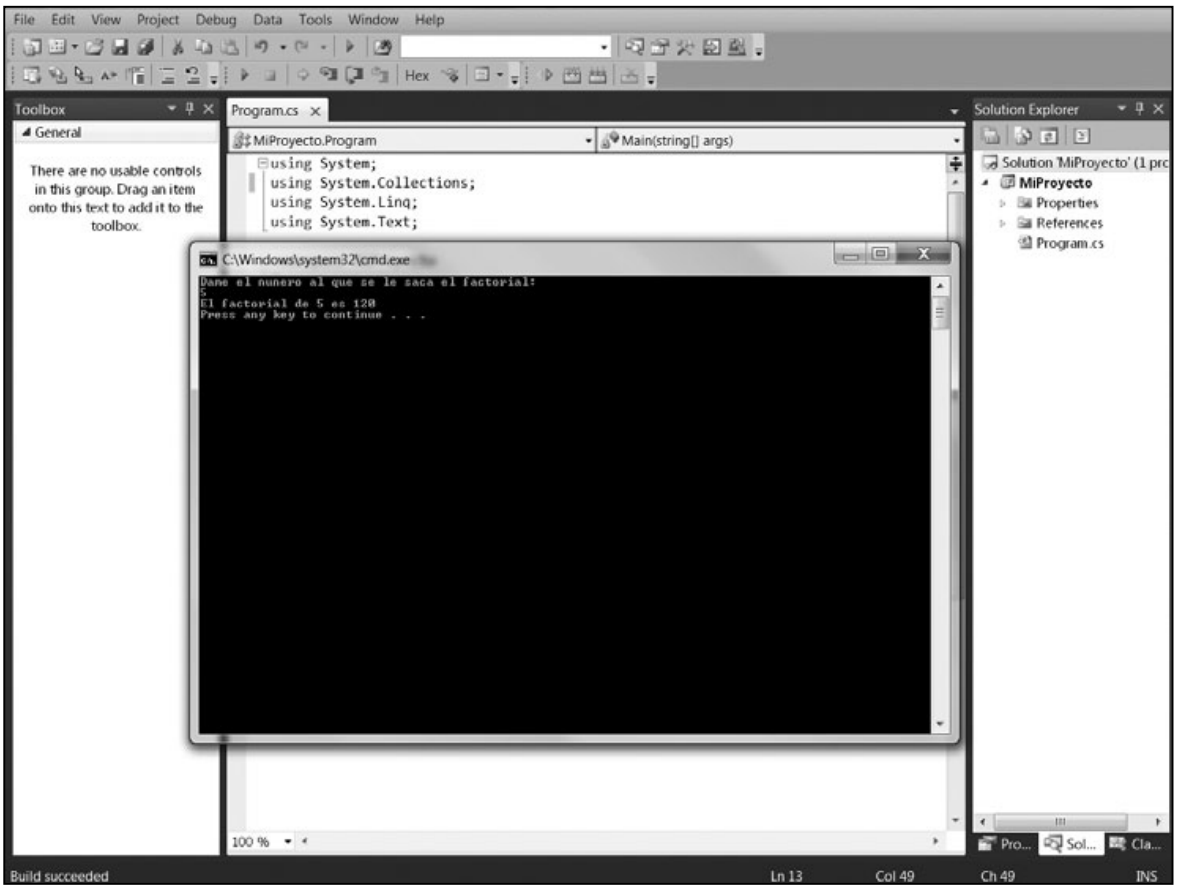

Figura 15. *El factorial del número es calculado adecuadamente.*

El ciclo do while

Hemos visto que el ciclo **for** es muy útil para repetir algo cuando sabemos el número de repeticiones previamente. Sin embargo, algunas veces no es posible saber el número de repeticiones que tendremos. Pensemos que una carpintería nos ha contratado para hacer un programa que transforme de pies y pulgadas a centímetros, ya que ellos lo usan para calcular el tamaño de las tablas. El programa es muy sencillo, pero no podemos dejarlo así ya que sería muy incómodo para ellos ejecutar el programa cada vez que necesitan hacer un corte.

Pensamos que esto se puede resolver con un ciclo **for**. ¿De cuántas vueltas el ciclo? Si colocamos **10** vueltas podemos pensar que es suficiente. Pero hay días que necesitan **15** conversiones y es necesario ejecutar el programa **dos** veces. Y los días que sólo necesitan **5** conversiones tienen que escribir **5** más tan sólo para finalizar el programa.

Podemos deducir del problema que efectivamente necesitamos un ciclo. Sin embargo, no sabemos el número de repeticiones previamente, por lo que el ciclo **for** no es adecuado. Necesitamos un ciclo que pueda ser controlado por una condición, y la evaluación de esta condición dependerá del estado del programa en un momento dado.

El ciclo **do while** nos permite hacer esto. Permite que cierto código se repita mientras una condición se evalúe como verdadera. El valor de la evaluación dependerá del estatus del programa en un momento dado.

El ciclo **do while** se codifica de la siguiente manera:

```
do  {
     Código
}(condición);
```

Si observamos el diagrama de flujo de la figura 16, podemos ver el funcionamiento interno del ciclo **do while** de una manera simple y clara.

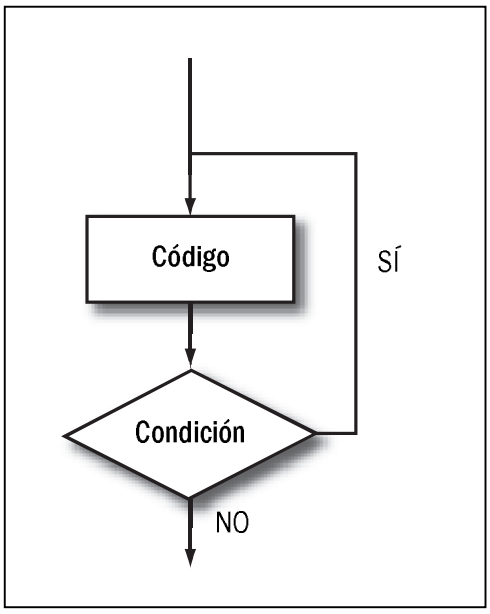

Figura 16. *Éste es el diagrama del ciclo do while. Podemos observar su comportamiento interno.*

Empecemos a recorrer el ciclo. En primer lugar encontramos el código que hay que llevar a cabo. Este código es definido adentro de un bloque de código aunque sea solamente una sentencia. Generalmente, en el código se modificará de alguna forma el valor de la variable o las variables que usamos para la expresión en la condición. Esta modificación de valor puede llevarse a cabo por medio de un cálculo, proceso o incluso una petición al usuario. Después tenemos la condición a evaluar. En la condición colocamos una expresión lógica o relacional. Si al evaluarse la condición obtenemos el valor **true**, entonces se repite el código. En caso de que la condición del valor de **false** ya no se repite y se continúa con el programa. Hay que tener en cuenta que a veces sucede que la condición se evalúa como falsa desde la primera vez. En este caso, solamente se habrá ejecutado el código una vez.

Veamos un ejemplo. Crearemos el programa para la carpintería y usaremos el ciclo **do while** para que el programa se repita el número de veces necesarias, aun sin saber cuántas veces son. Para lograr esto tenemos que pensar bien en nuestra condición y cuál es el estado del programa que evaluaremos en ella.

La forma más sencilla de hacerlo es preguntarle al usuario si desea hacer otra conversión. Si lo afirma, se repite el código para convertir. En caso de que el usuario no lo desee, el programa finaliza. De esta forma, es controlada la repetición del ciclo pudiéndola repetir, aun sin saber cuántas veces hay que hacerlo.
Nuestro programa tendrá el siguiente diagrama de flujo.

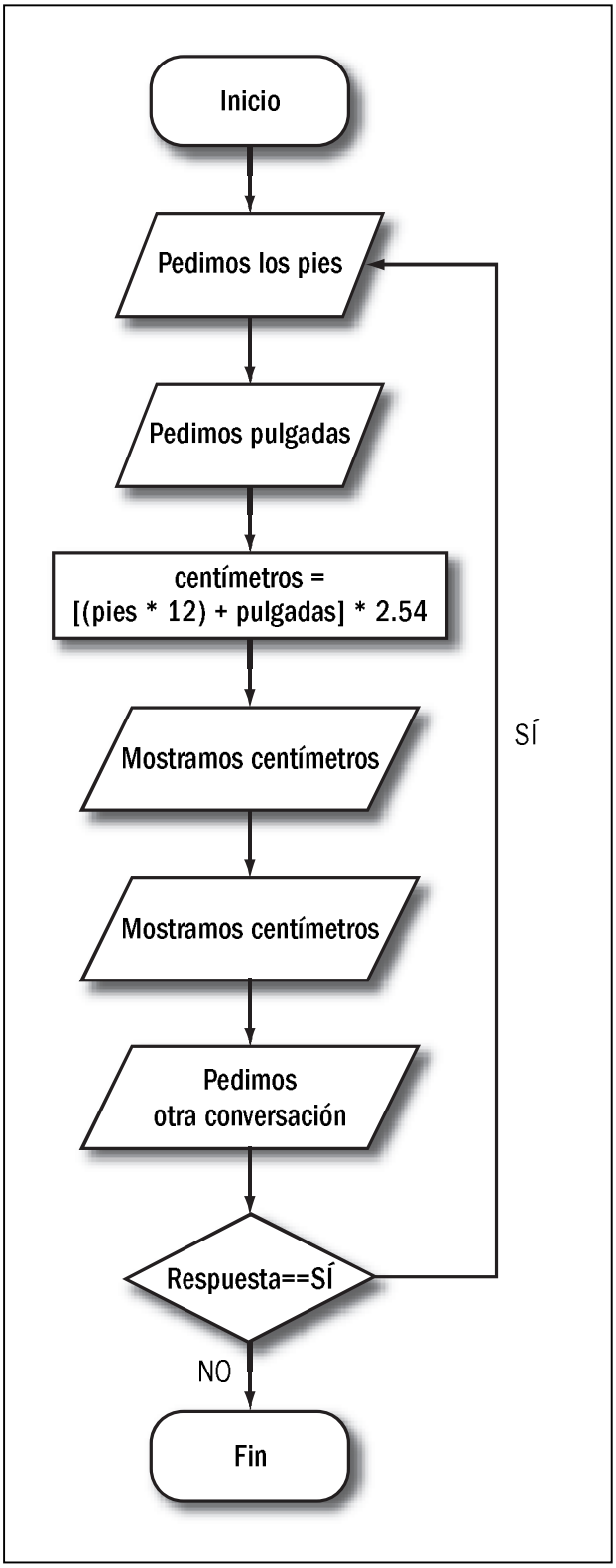

Figura 17. *Aquí podemos observar el programar y distinguir el ciclo* ***do while.***

El programa quedará de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            float pies = 0.0f;  // Cantidad de pies
            float pulgadas = 0.0f;  // Cantidad de pulgadas
            float centimetros = 0.0f;  // Resultado en centímetros
            string respuesta = "";  // Respuesta para otro cálculo
            string valor = "";

            do
            {
                // Pedimos los pies
                Console.WriteLine("Cuántos pies:");
                valor = Console.ReadLine();
                pies = Convert.ToSingle(valor);

                // Pedimos las pulgadas
                Console.WriteLine("Cuántas pulgadas:");
                valor = Console.ReadLine();
                pulgadas = Convert.ToSingle(valor);

                // Convertimos a centimetros
                centímetros = ((pies * 12) + pulgadas) * 2.54f;

                // Mostramos el resultado
               Console.WriteLine("Son {0} centímetros", centimetros);

                // Preguntamos si otra conversión
```

```
                Console.WriteLine("Deseas hacer otra conversión
                      (si/no)?");
                respuesta = Console.ReadLine();

            } while (respuesta == "si");

        }
    }
}
```

Ejecutemos el programa.

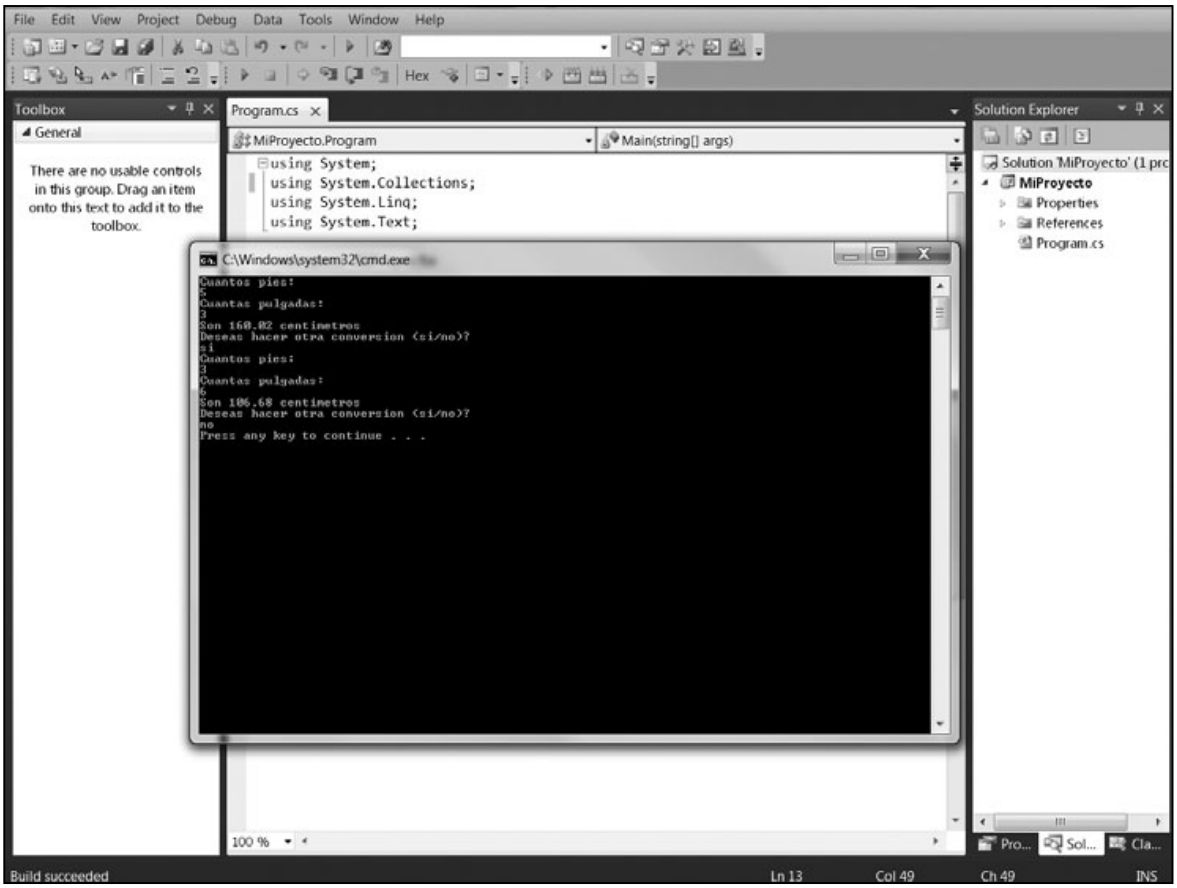

Figura 18. *En la ejecución podemos verificar que tenemos un ciclo, pero puede terminar cuando lo deseemos.*

Podemos observar que hemos colocado adentro del ciclo la parte de conversión de pies y pulgadas a centímetros. Después de mostrar el resultado le preguntamos al usuario si desea realizar otra conversión. En caso afirmativo, repetimos el ciclo. Si el usuario no desea hacer otra conversión, el ciclo finaliza. Esto nos permite que a veces se repita cinco veces, cuando sea necesario diez, y así dependiendo de las necesidades del usuario. Hemos resuelto el problema.

El ciclo **do while** nos da flexibilidad extra, pero siempre es necesario usar el tipo de ciclo adecuado al problema que tenemos.
Veamos otro ejemplo donde podemos utilizar este ciclo. En uno de los programas anteriores le preguntábamos al usuario qué operación deseaba realizar y luego los operandos. El programa tal y como está solamente se ejecuta una vez, pero si usamos el ciclo **do while** y colocamos una nueva opción en el menú, entonces el usuario puede realizar las operaciones que sean necesarias o solamente una como se usaba anteriormente. Este tipo de comportamiento es muy útil y fácil de implementar con el ciclo **do while.**
Si modificamos el programa quedará de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            float a = 0.0f;
            float b = 0.0f;
            float resultado = 0.0f;
            string valor = "";
            int opcion = 0;

            // Tenemos el ciclo
            do
            {
                // Mostramos el menú
                Console.WriteLine("1- Suma");
                Console.WriteLine("2- Resta");
                Console.WriteLine("3- División");
                Console.WriteLine("4- Multiplicación");
                Console.WriteLine("5- Salir");
```

```
Console.Write("Que operación deseas hacer: ");
valor = Console.ReadLine();
opcion = Convert.ToInt32(valor);

if (opcion != 5)
{
    // Pedimos el primer número
    Console.Write("Dame el primer número:");
    valor = Console.ReadLine();
    a = Convert.ToSingle(valor);

    // Pedimos el segundo número
    Console.Write("Dame el segundo número:");
    valor = Console.ReadLine();
    b = Convert.ToSingle(valor);

    switch (opcion)
    {
        // Verificamos para suma
        case 1:
            resultado = a + b;
            break;

        // Verificamos para resta
        case 2:
            resultado = a - b;
            break;

        // Verificamos para división
        case 3:
            if (b != 0) // este if esta anidado
                resultado = a / b;
           else // Este else pertenece al segundo if
           Console.WriteLine("Divisor no válido");
            break;

        // Verificamos para la multiplicación
        case 4:
            resultado = a * b;
            break;
```

```
                        // Si no se cumple ninguno de los casos
                            anteriores
                        default:
                          Console.WriteLine("Opción no válida");
                            break;

                    }

                    // Mostramos el resultado
                Console.WriteLine("El resultado es: {0}",
                          resultado);
                 }
            } while (opcion != 5);

        }
    }
}
```

El ciclo while

Habiendo comprendido el ciclo **Do While**, estamos en condiciones de ver otro tipo de ciclo. Este ciclo se conoce como **while** y en cierta forma se asemeja al anterior, pero tiene sus propias características que debemos conocer para utilizarlo correctamente. El ciclo **while** también puede ser utilizado cuando tenemos algo que se debe repetir pero no conocemos el número de repeticiones previamente. La repetición del ciclo tiene que ver con el cumplimiento de una condición. A diferencia del ciclo **do while**, este ciclo puede no ejecutarse ni siquiera una vez. Su estructura es la siguiente:

```
while(condicion) {
     Código
}
```

En el programa de las operaciones matemáticas hemos agregado un ciclo **do while**. Sin embargo, nuestro trabajo no termina ahí. Tenemos que ejecutarlo para ver si se comporta de la forma deseada. Por eso fue necesario que adicionáramos un **if**, ya que de lo contrario nos pediría los operadores aun cuando lo que deseáramos fuera salir.

Para comprender cómo funciona, veamos el diagrama de flujo.

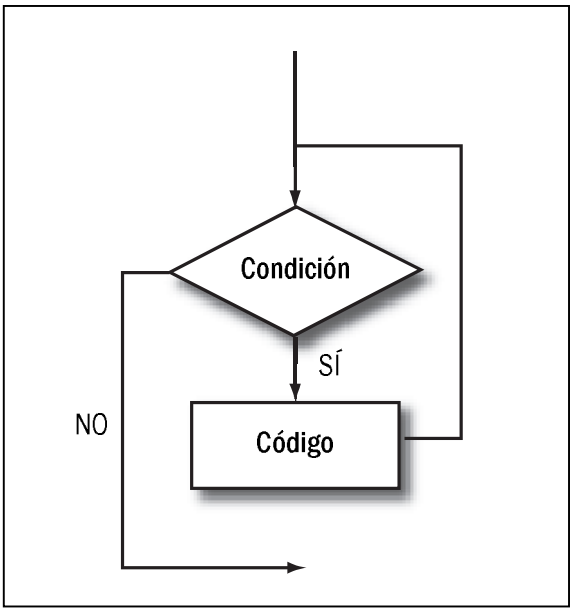

Figura 19. *Este ciclo inicia con una condición y esto lo debemos de tener en cuenta en nuestros algoritmos.*

Si observamos el diagrama de flujo lo primero que encontramos es una condición. Adentro de esta condición colocaremos una expresión lógica o relacional. Si la condición se evalúa como verdadera, entonces se procede a ejecutar el código. Después del código se regresa a la condición. Si la condición no se cumple, entonces no se ejecuta el código y se continúa con el resto del programa.

El tener la condición al inicio del ciclo nos lleva a un punto importante. Si la condición no se cumple desde el inicio, entonces el ciclo nunca se lleva a cabo. Si nuestro algoritmo está bien diseñado, esto es deseable.

Ahora podemos hacer un ejemplo de programa que utilice el ciclo **while**. En este ejemplo utilizaremos la característica del ciclo que puede repetirse o no repetirse ni siquiera una vez. Imaginemos que tenemos que hacer un programa de control para enfriar una caldera. La caldera debe ser enfriada a 20 grados centígrados.

El ciclo **while** será usado para reducir la temperatura de uno en uno para cada vuelta del ciclo hasta que lleguemos a 20 grados centígrados. La ventaja que nos da este ciclo

Algunos programadores usan el ciclo **while** y para garantizar su ejecución les asignan valores a las variables de forma tal que la condición se cumpla al inicio. Esto es recomendable en pocos casos. Si necesitamos un ciclo que garantice la entrada es mejor hacer uso de **do while** en lugar de forzar nuestra lógica con **while**.

es que si la temperatura es menor a 20 grados, ni siquiera se entra al ciclo y no se lleva a cabo ningún enfriamiento. En este caso, aprovechamos las propiedades del ciclo. Veamos el código de este programa:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AplicacionBase
{
    class Program
    {
        // Esta es la función principal del programa
        // Aquí inicia la aplicación
        static void Main(string[] args)
        {
            // Variables necesarias
            int temperatura = 0;
            string valor = "";

            // Pedimos la temperatura
            Console.WriteLine("Dame la temperatura actual:");
            valor = Console.ReadLine();

            temperatura = Convert.ToInt32(valor);

            // El ciclo reduce la temperatura
            while (temperatura > 20)
            {
                // Disminuimos la temperatura
                temperatura--;
```

Al igual que con los **if**, es posible tener ciclos anidados. Como adentro de los ciclos podemos colocar cualquier código válido, entonces también podemos colocar otro ciclo. Cuando un ciclo se encuentra adentro de otro, decimos que están anidados. Hay que tener mucho cuidado con las variables de control y las condiciones para evitar problemas de lógica.

```
                    Console.WriteLine("Temperatura->{0}",
                            temperatura);
                }

                // Mostramos la temperatura final

            Console.WriteLine("La temperatura final es {0}",
                temperatura);
             }
        }
    }
```

Para comprobar que efectivamente el ciclo actúa como hemos dicho, debemos correr el programa dos veces. La primera vez colocaremos la temperatura en valor mayor a 20°C. Esto debe hacer que el ciclo se lleve a cabo. La segunda vez colocaremos un valor menor que 20. En este caso, el programa no deberá entrar al ciclo.

Veamos el resultado de la primera ejecución:

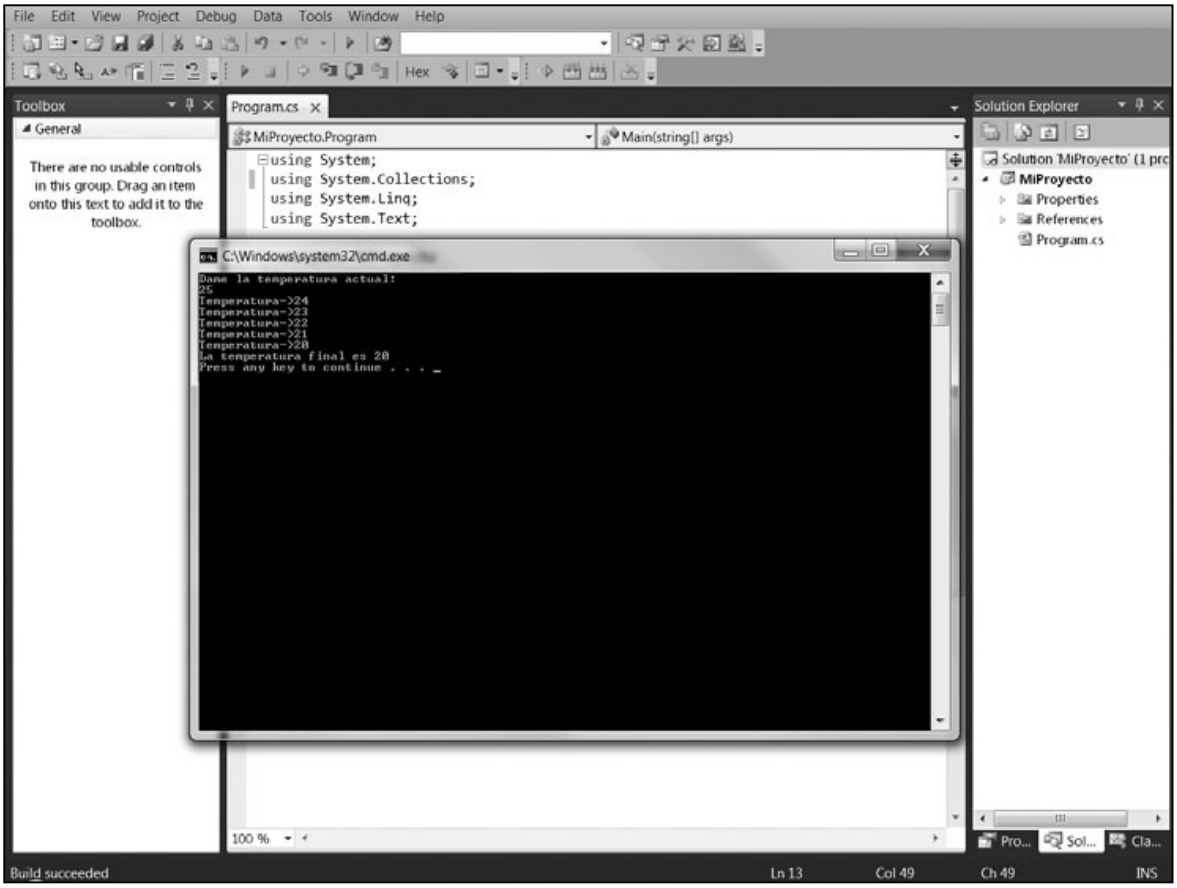

Figura 20. *Podemos observar que se ha entrado al ciclo y que se lleva a cabo mientras la temperatura es mayor a 20°.*

Si llevamos a cabo la segunda ejecución obtenemos lo siguiente:

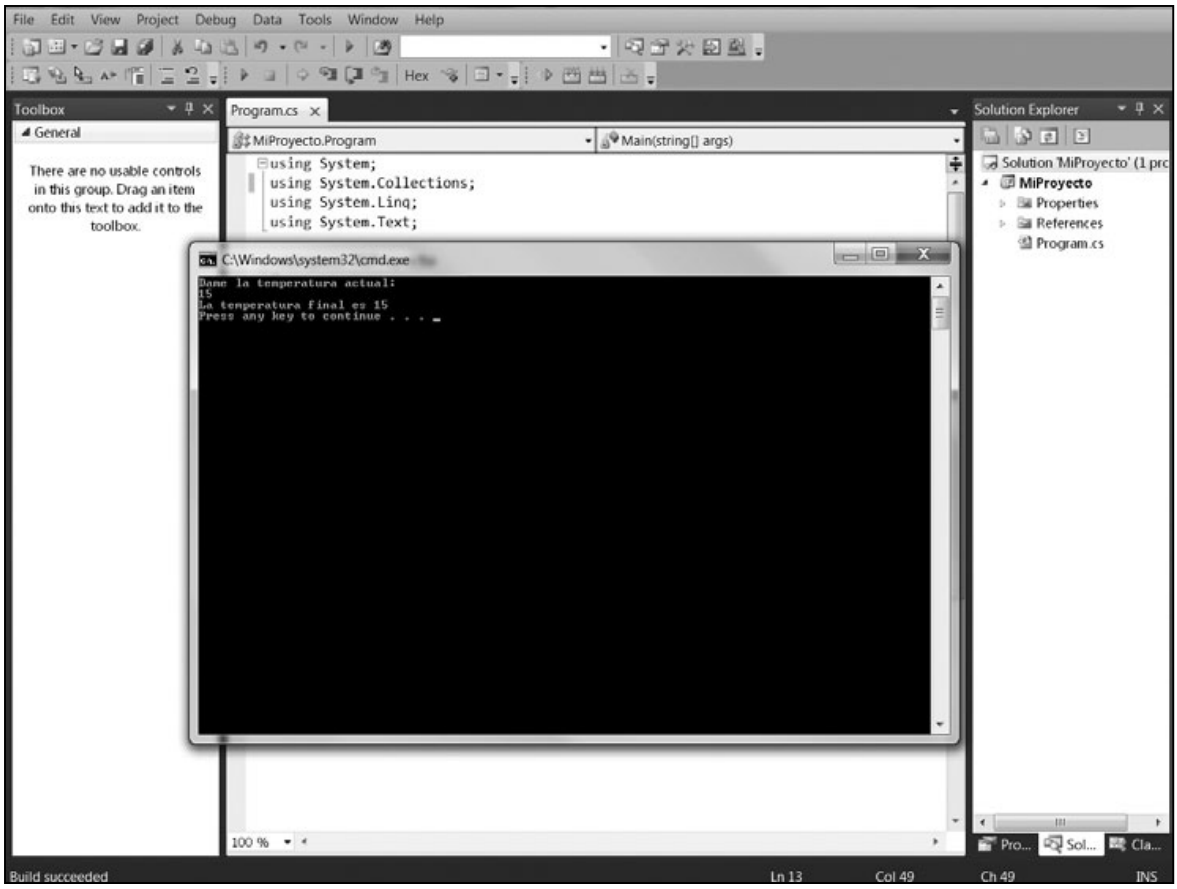

Figura 21. *En este caso no se entra al ciclo en la ejecución del programa.*

Con esto hemos visto los tres ciclos principales disponibles en C# y de qué forma podemos utilizarlos dentro de nuestros desarrollos.

RESUMEN

Los ciclos nos permiten repetir la ejecución de cierto código. El ciclo for es usado para repetir algo un número determinado de veces. Este ciclo necesita de una variable de control con valor inicial, una condición y un incremento. El ciclo do while nos permite repetir cierto código un número de veces desconocido y se ejecuta al menos una vez. El número de veces depende de una condición. El ciclo while también nos permite repetir el código un número desconocido de veces, pero en este caso el ciclo puede o no puede ejecutarse dependiendo de su condición.